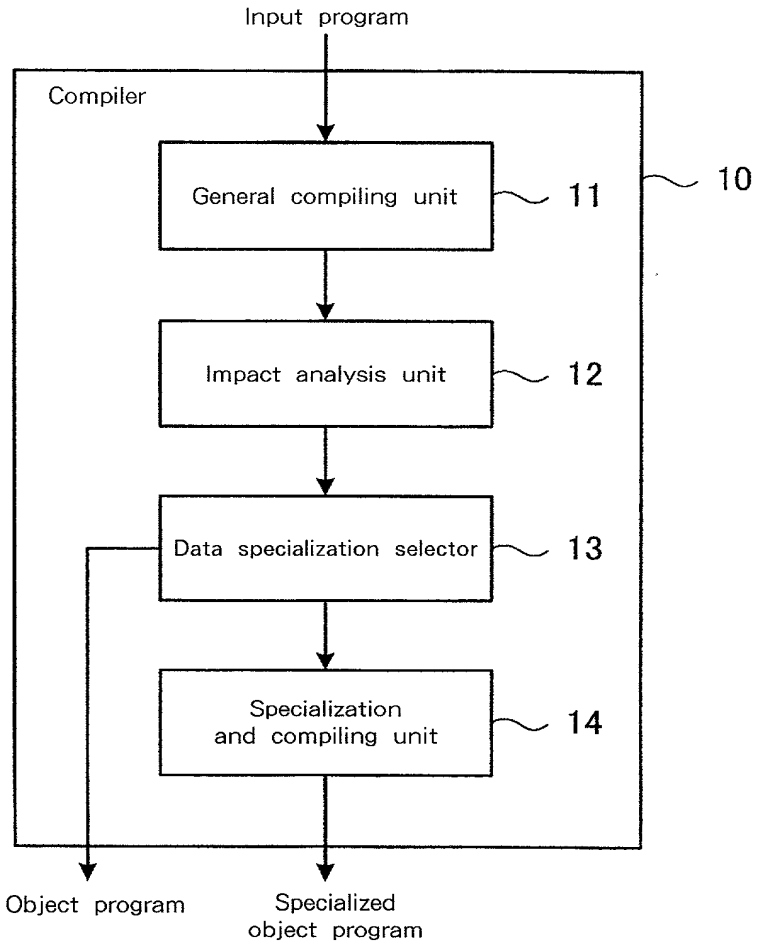
Input program
Compiler
General compiling unit
11
10
Impact analysis unit
12
Data specialization selector
13
Specialization
and compiling unit
14
Object program
Specialized
object program

Fig. 1

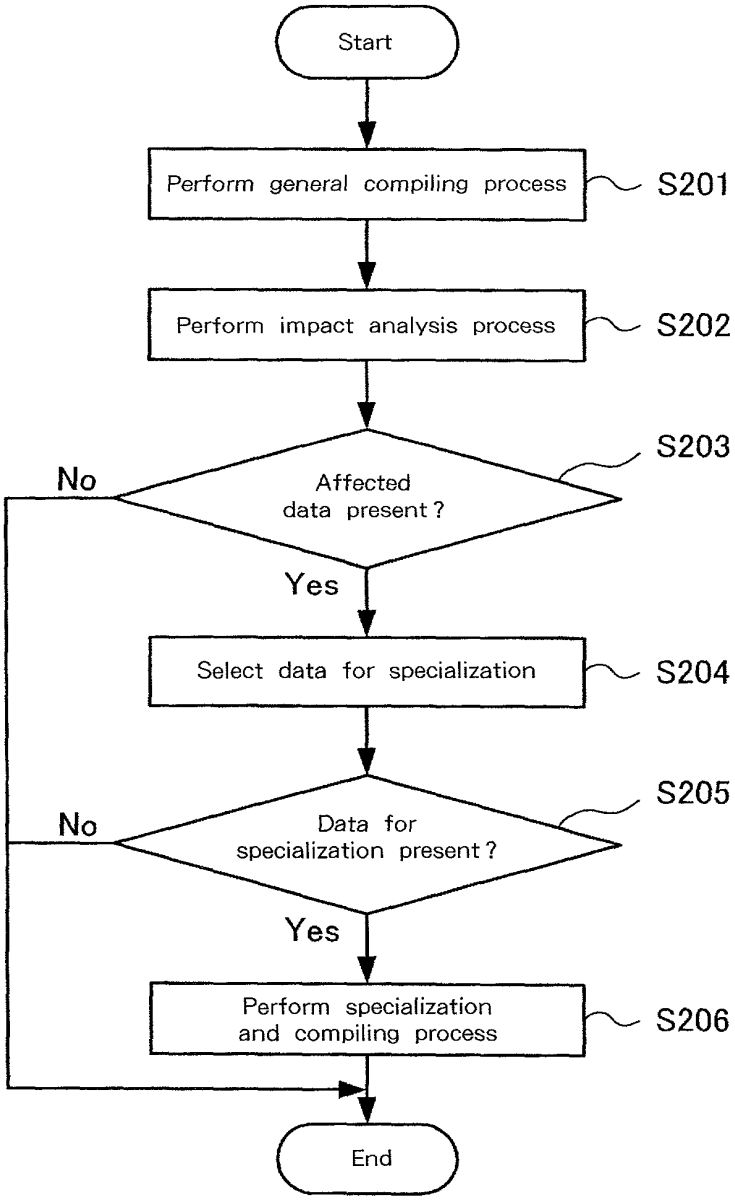

Fig. 2

Type	Command
Constant	Four fundamental arithmetic calculations Comparison command for a variable other than an object (including a "switch" command)
Not Null	General commands for clearly determining whether the value of a variable is null
Designation of a class	INSTANCEOF CHECKCAST General virtual method call commands
Class other than an array	General commands for determining whether an object is an array: specifically INSTANCEOF CHECKCAST INVOKEVIRTUALOBJECT_QUICK

Fig. 3

```
Effective = φ;
for (each p ∈ all the parameters ) {
    for (each type ∈ ALLTYPE) {
        Impact[p][type] = 0;
    }

    Estimate(impact, p, p, φ);

    for (each type ∈ ALLTYPE) {
        if (Impact[p][type] > threshold value A ) {/* equivalent to guard code when a threshold value is 1 */
            Effective U= { p, type };
        }
    }
}
```

Fig. 4

```
Estimate(impact, v, p, type)
{
   for (each l ∈ command using v ) {          /* use DU-chain */
       if (only one definition of v in command "l" ) {  /* use UD-chain */
           tmp_type = which specialization type of those in ALLTYPE should be used to optimize command l using v ; – (1)
           if (tmp_type != φ) { /* if command l is optimized using v */
               if (type == φ) {
                   type = tmp_type;
               }
               Impact[p][type] += impact value of optimized command l when the cost of guard code relative to
               type is defined as 1 ; /* for example, effects are obtained twice of guard code when impact value is 2,
               or effects are obtained half of guard code when impact value is 0.5. */
               if ( through specialization, command l is a command for substitution into a constant ) {
                   Estimate(impact, local variable into which results of l are inserted, p, type) ;
               }
           }
       }
   }
}
```

Fig. 5

```
if (Effective == φ) halt parameter specialization process
for (each { p, type } ∈ Effective) {
      obtain statistical data for parameter p under condition of specialization method "type".
}
```

Fig. 6

```
Specialize = φ;
for (each { p, type } ∈ Effective) {
    odds = Highest probability among statistical data for {p, type} ;
    val = value of {p, type} corresponding to odds;
    if (Impact[p][type] * odds > threshold value B) {  /* equivalent to guard code when a threshold value is 1 */
       Specialize ∪= { p, type, val };
    }
}
if (Specialize == φ) halt parameter specialization process
```

Fig. 7

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcBegin < 0) {
        throw new StringIndexOutOfBoundsException(srcBegin);
    }
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (srcBegin > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd - srcBegin);
    }
    System.arraycopy(this.value, this.offset + srcBegin, dst, dstBegin,
                     srcEnd - srcBegin);
}
```

Fig. 8

(A) Impact [p] [type] when impact analysis is completed

p	type	Impact [p] [type]
srcBegin	Constant	2.25
	Others	0
srcEnd	Constant	1.25
	Others	0
dst	All	0
dstBegin	All	0

(B)

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcBegin < 0) {
        throw new StringIndexOutOfBoundsException(srcBegin);
    }
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (srcBegin > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd - srcBegin);
    }
    System.arraycopy(this.value, this.offset + srcBegin, dst, dstBegin,
                     srcEnd - srcBegin);
}
```

Fig. 9

Statistics when Jack benchmark of SPECjvm98 is employed

srcBegin	Value 0	Probability 100 %
srcEnd	Value 1	Probability 68 %
	Value 0	Probability 15 %
	Value 2	Probability 4 %
	Value 3	Probability 3 %
	Value 4	Probability 3 %
	(not shown)	

Fig. 10

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (0 > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    System.arraycopy(this.value, this.offset, dst, dstBegin, srcEnd);
}
```

Fig. 11

```
boolean dispatchEvent(AWTEvent e) {
   boolean ret = false;
   if ((e instanceof MouseEvent) &&
       ((eventMask & MOUSE_MASK) != 0)) {
       MouseEvent me = (MouseEvent) e;
       ret = processMouseEvent(me);
   } else if (e instanceof FocusEvent) {
       FocusEvent fe = (FocusEvent) e;
       ret = processFocusEvent(fe);
   } else if (e instanceof KeyEvent) {
       KeyEvent ke = (KeyEvent) e;
       ret = processKeyEvent(ke);
   }
   return ret;
}
```

Fig. 12

```
boolean dispatchEvent(AWTEvent e) {
   boolean ret = false;
   if ((e instanceof MouseEvent) &&
       ((eventMask & MOUSE_MASK) != 0)) {
       MouseEvent me = e;       /* checkcast has been removed */
       ret = processMouseEvent(me);
   } else if (e instanceof FocusEvent) {
       FocusEvent fe = e;       /* checkcast has been removed */
       ret = processFocusEvent(fe);
   } else if (e instanceof KeyEvent) {
       KeyEvent ke = e;  /* checkcast has been removed */
       ret = processKeyEvent(ke);
   }
   return ret;
}
```

Fig. 13

Impact [p] [type] when impact analysis is completed

p	type	Impact [p] [type]
e	Designation of a class	3.94
	Other	0

Fig. 14

```
boolean dispatchEvent(AWTEvent e) {
    boolean ret = false;
    if (e != null) {
       KeyEvent ke = e;
       ret = processKeyEvent(ke);
    }
    return ret;
}
```

Fig. 15

```
boolean dispatchEvent(AWTEvent e) {
    boolean ret;
    KeyEvent ke = e;
    ret = processKeyEvent(ke);
    return ret;
}
```

Fig. 16

```
Effective = φ;
ALLTYPE = all specialization types
for (each p ∈ DownSafety [head of a method]) {
    Ivar = local variable including results of p;
    for (each type ∈ ALLTYPE) {
        Impact[p][type] = 0;
    }

    Estimate(impact, Ivar, Ivar, φ);

    for (each type ∈ ALLTYPE) {
        if (Impact[p][type] > threshold value A ) { /* equivalent to guard code when a threshold value is 1 */
            Effective ∪= { p, type };
        }
    }
}
```

Fig. 17

```
public int indexOf(int ch, int fromIndex) {
    int max = this.offset + this.count;
    char v[] = this.value;

    if (fromIndex < 0) {
        fromIndex = 0;
    } else if (fromIndex >= this.count) {
        return -1;
    }
    for (int i = this.offset + fromIndex ; i < max ; i++) {
        if (v[i] == ch) {
            return i - this.offset;
        }
    }
    return -1;
}
```

Fig. 18

```
public int indexOf(int ch, int fromIndex) {
    int offset = this.offset;
    int count = this.count;
    int max = offset + count;
    char v[] = this.value;

    if (fromIndex < 0) {
        fromIndex = 0;
    } else if (fromIndex >= count) {
        return -1;
    }
    for (int i = offset + fromIndex ; i < max ; i++) {  -- (1)
        if (v[i] == ch) {
            return i - offset;
        }
    }
    return -1;
}
```

Fig. 19

Impact [p] [type] when impact analysis is completed

p	type	Impact [p] [type]
ch	All	0
fromIndex	Constant	1.75
	Others	0
this.offset	Constant	5.75
	Others	0
this.count	Constant	5.75
	Others	0
this.value	All	0

Fig. 20

```
public int indexOf(int ch, int
fromIndex) {
    char v[] = this.value;

    if (v[0] == ch) return 0;
    if (v[1] == ch) return 1;
    if (v[2] == ch) return 2;
    if (v[3] == ch) return 3;
    return -1;
}
```

Fig. 21